L7: Entry 1 of 1 File: USPT Jun 24, 2003

DOCUMENT-IDENTIFIER: US 6584493 B1
**** See image for Certificate of Correction ****
TITLE: Multiparty conferencing and collaboration system utilizing a per-host model command, control and communication structure

Abstract Text (1):
A networking conferencing and collaboration tool utilizing an enhanced T.128 application sharing protocol. This enhanced protocol is based on a per-host model command, control, and communication structure. This per-host model reduces network traffic, allows greater scalability through dynamic system resource allocation, allows a single host to establish and maintain a share session with no other members present, and supports true color graphics. The per-host model allows private communication between the host and a remote with periodic broadcasts of updates by the host to the entire share group. This per-host model also allows the host to allow, revoke, pause, and invite control of the shared applications. Subsequent passing of control is provided, also with the hosts acceptance. The model contains no fixed limit on the number of participants, and dynamically allocates resources when needed to share or control a shared application. These resources are then freed when no longer needed. Calculation of minimum capabilities is conducted by the host as the membership of the share changes. The host then transmits these requirements to the share group.

Application Filing Date (1):
19990914

Brief Summary Text (9):
The T.128 model utilized in this NetMeeting.TM. system was a global free-for-all model where all members were peers. Every person in the conference would maintain a global list, ordered from front to back of all of the shared applications of everybody in the conference, merged completely together. Each person had to be in lock-step with the others, so that the positions, order, and appearance of all shared applications were in sync. Any change in order, state, or position had to be transmitted to everyone in the meeting. This would frequently cause a Ping-Pong effect whereby the new global list would be sent, someone in the conference would decide it was out of date because their shared application had moved, and would transmit a new window list, and so forth. This problem was exacerbated by the collaboration model which also required that all members of the conference periodically broadcast his or her mouse position. This additional network traffic was necessitated by the toggle during collaboration whereby only one of the collaborators controlled all of the collaborators' mice and keyboards. Therefore, the host would periodically need to check where everyone's cursor is positioned.

Brief Summary Text (18):
By implementing a per-host model whereby communication with and control of the host takes place in a private fashion between the host and a remote with periodic broadcast updates by the host to the entire share group, the total number of network messages which are required to be transmitted between the members of the share group are greatly reduced. To contrast this per-host model, the prior

versions of NetMeeting.TM. utilized a global model where each person in the conference would maintain a global list, ordering front to back, of all the shared applications of everybody in the conference, merged completely together. This resulted not only in a large number of messages being initially required to maintain the members of the conference in lockstep, but also had a ripple effect whereby each adjustment resulting from the reception of such a message would essentially be echoed back in a broadcast global fashion since a change had now occurred on that user's system.

Brief Summary Text (19):
In the per-host model, the network traffic almost always originates from the host only as opposed to being globally transmitted by each of the members of the conference. While multiple members of the conference may share an application, this per-host model allows each of those members who are sharing to act like a miniature server for the conference, i.e. a host of that shared application. Updates, therefore, instead of being globally transmitted by all members of the conference, simply now stream down from the host. This requires only that the members of the conference need the capabilities of that particular host. A performance improvement is particularly noticeable in the per-host model when a member of the conference is in control of the host. In the per-host model the controlling member transmits its keyboard and mouse move messages privately to the host who then periodically broadcasts the current cursor position to all members of the conference.

Brief Summary Text (21):
At a detailed level, the following packet changes are implemented for NetMeeting.TM. 3.0: fewer and smaller Shared Window Lists (SWL) broadcast packets; fewer Active Window Coordinator (AWC) broadcast packets; fewer cursor broadcast packets; fewer Host Entity Tracker (HET) broadcast packets; fewer Synchronizing New Individuals (SNI) broadcast packets; fewer control arbitration (CA) packets; new control (CA30) packets; and fewer Input Manager (IM) packets. These revised and new packets, and the way that they are shared when a new member joins the conference, result in a significant reduction in traffic during application sharing just to get everyone in synchronism with the sharing application.

Detailed Description Text (13):
These enhancements are embodied in a new T.128 model. Unlike the global, chaotic model of NetMeeting.TM. 2.0, the T.128 model of the instant invention is a per-host model. In this model, each person hosting (sharing an application) acts like a miniature server for the conference. Network traffic almost always originates from hosts only, and wends its way down to the others viewing in the conference. Members have separate state information for each person who is sharing. The updates (shared application lists, the graphics of the shared applications, the current cursor position and shape) stream down from the host. To interpret packets coming from the host only requires the viewer to know the capabilities of the host. The network traffic from a viewer is not broadcast, but sent privately back to the host, when controlling. The rest of the members in the conference see the results (changes in appearance, movement of the cursor, etc.) which are broadcast by the host later. Since input, mouse and keyboard messages, are only sent privately from a controller to a host, the latency and responsiveness, especially of the mouse, is much improved. This is especially noticeable in large conferences since there is no performance penalty as more people participate.

Detailed Description Text (14):
Specific changes in the T.128 protocol contemplated herein which allow the realization of the advantages of the instant invention involve not sending ignored user name/capabilities in some control packets; removing/ignoring many capabilities; streamlining cache capabilities negotiation when a node starts to share, stops sharing, and when a new person joins the conference; creating caches; order encoding data each time a note starts to host, and freeing/cleanup when the node stops hosting; eliminating some packet broadcasts to everyone in the

conference or replacing them with targeted sends to an individual; and a new control model and new control packets. Like file transfer (T.127), T.128 has distinct send side (hosting) and receive side (viewing) parts that compose the logical T.128 applet. In the instant invention, a node in a conference is a host when it is sharing applications or its desktop. The act of hosting or sharing is the process of trapping the graphics on the screen and transmitting the updates for the entities that are shared. There is an UI applet for hosting which basically is a dialog that lists all of the top level applications running along with the entire desktop. This UI shows what is shared, and allows a user to share/unshare items and to stop sharing everything. This applet also allows a user to change whether the shared applications/desktop are controllable, and has other options for 24-bit color sharing and automatic handling of control requests. A node in a conference is a viewer when a remote node is hosting and it has AS active (unless application sharing is prevented by system policy). The UI for viewing may be a frame window displaying the shared contents of the remote host.

Detailed Description Text (17):
In an exemplary embodiment of the instant invention, there are four parts to the T.128 protocol: CMG, the T.120 wiring to find out about calls and activate AS sessions; S20, the share establishment/capabilities exchange/member join/leave/share termination part; CPC, the member capabilities data sent via S20 control packets; and DATA, the AS streaming part, for hosting and controlling, which accounts for the capabilities of the share members so it does not send data or packets that they cannot understand. The following description of an exemplary embodiment is included by way of example, and not by way of limitation, to enable one of ordinary skill in the art to practice an embodiment of the instant invention.

Detailed Description Text (18):
CMG is utilized to find out when calls start, when calls end, and when members are added/removed. When a new call is starting, AS receives a permit to enroll indication notification. It enrolls its application by filling in a GCC session key field with appropriate key types, capability structure, etc. as is standard. The enroll request is filled in and the application enroll method is called to enroll or unenroll in the conference starting/ending. When an enroll confirm notification is received, it looks for success or failure. If failure, it cleans up as though the call had ended. If success, it enrolls the new member and watches for the application roster report indication notifications that indicate change. Finally, CMG looks for the local node in the member list, the first time it is seen the member is considered finally to be in a T.120 call. It processes its own section, looking for new members so it can add them, and for old members now gone so it can remove them. These are GROUPWARE members only, application sharing (AS) member addition/removal comes independently through the S20 protocol. When AS finally believes itself to be in a T.120 call, it attaches to the MCS domain so it can send and receive data. When the domain attachment is confirmed, AS joins two MCS channels: its own MCS user channel; and the MCS AS broadcast channel. When both of the MCS channels are joined, the system is ready.

Detailed Description Text (20):
When the MCS channels are joined after a new call starts, AS broadcasts an S20_JOIN packet on the T.128 channel to join an existing share. If there is an existing share, the system will see a response on the T.128 channel, an S20_RESPOND packet with the MCS user ID as the node being responded to, from the node that created the share. Others in the share will see the S20_RESPOND also. Each will add the new member into their member lists, and if successful, each will broadcast another S20_RESPOND out on the T.128 channel. This will be ignored by the existing people in the share, since they already know about these other existing members. The remote people are added one by one this way into the new member's member list. Preferably, these messages acknowledging the new person and informing him about the existing members are only sent privately to the new member to reduce network

traffic. The capabilities and user name are a couple hundred bytes of data and are included in these packets. When leaving an existing share, a node broadcasts an S20_LEAVE packet on the T.128 channel. The other members of the share will see this message and remove the member leaving from their member lists. When something goes wrong letting a new node join into a share, e.g. if a share is taking place with members utilizing NetMeeting.TM. 3.0 embodying the teachings of the instant invention and a member utilizing the old NetMeeting.TM. 2.x tries to join, the share creator broadcasts an S20_DELETE packet on the T.128 channel with the MCS user ID of the node being ejected. The share ends when the share creator leaves the share. The share creator will broadcast an S20_END packet on the T.128 channel, and the other members will clean up and terminate also.

Detailed Description Text (22):
For CPC, the member name and the member application sharing capabilities are exchanged via the S20 control packets. When a new member is added into a share list, the share capabilities are recalculated. Most of these capabilities are used to determine what kind of application sharing data to send when hosting. Prior systems used to do a lot of recalculation whether it was hosting or not. The system of the instant invention, however, only does recalculation when it is hosting, since starting to host will calculate the capabilities based off the people in the share at the time anyway. Moreover, a lot of recalculation is gone completely making it a lot faster and easier to handle a new member of the share. The total capabilities block is basically a list of the area capabilities (PROTCAPS) blocks with IDs for each. Members ignore PROTCAPS blocks with unrecognized IDs.

Detailed Description Text (25):
The data packets transmitted, as indicated above, are prefixed with the S20DATAPACKET structures, then a DATAPACKETHEADER. These are the stream types (dictionaries). All of the packet types are grouped into one of the few streams, the groups have much in common as far as contents and sizes, so they compress together well. The types of data packets include drawing/graphics updates sent by the host and the supported font list, which is sent by everyone in the conference to each other. While this is really a capability, it is so large (could be 32K) it is not exchanged via the S20 protocol. The data packets also include control packets, active window notification sent by the host, or activate/restore requests sent to the host, the shared window list sent by the host, the hosting state (hosting applications, hosting desktop, hosting nothing anymore) sent by the host, cursor appearance/position notification sent by the host, keyboard/mouse input sent from controller to the host, and a sync packet from existing member of a share to a new member to let the new member know that the group is aware of him and how to handle data from the group. This is needed because of cache states, order encoding data, and font lists. Further data packet include the changed capability, e.g. when desktop size/color depth changes. The compression types on the packet data include: not compressed; PKZIP compressed, but atomic, no info about previous packets is need to decompress; and persistent-PKZIP compressed, information about previous packets sent on same stream is needed to decompress.

Detailed Description Text (26):
As compared with the prior versions of NetMeeting.TM., the system of the instant invention provides fewer and smaller Shared-Window-List (SWL) broadcast packets. These packets provide notification of the list of shard window states, positions, sizes, along with areas that are obscured, if anything has changed since last time. In the prior systems, the SWL packets were sent by people hosting, and by non-hosts if they were in control so that everyone else would sync to the placement of shadows on their system. Shadow windows, representing shared information from other people, were always included in the list as well. When joining and leaving, an empty packet was always sent, even if the member was not hosting. There were lots of packets dropped due to collisions in z-order when multiple people were hosting, or packets applied and then another z-order packet would be sent out in a Ping-Pong effect. Now, only the host sends the SWL packets, the contents of which are simply

windows shared on the host, and things that obscure parts of the shared windows. Therefore, when not hosting and joining/leaving share, no packets are sent. Further, z-order is not changing all the time because a plurality of shadows are not changing independently as before.

Detailed Description Text (33):
In addition to the above, the system of the instant invention also provides fewer Input Manager (IM) packets. In prior systems, the person in control if collaborating, or any detached host, broadcasted input packets every mouse move/click/keypress. Everyone not collaborating treated these like notifications, and updates the keyboard state table/mouse position of the sender (and all controlled nodes if from person in control). Everyone collaborating treated these like requests and injected the input into their machine. This would be horribly slow in a large conference if a controller moved the mouse a lot. NetMeeting.TM. 2.x collapsed the mouse packets which resulted in jerky and unresponsive cursor movement. In the system of the instant invention, input packets are sent privately from a controller to a host, not broadcasted. The controlled host then periodically broadcasts the new cursor position/shape due to the input played back from the controller. This allows multiple hosts to be simultaneously controlled by multiple independent users. The result in very high mouse fidelity and lower bandwidth in a large conference.

Detailed Description Text (36):
On the viewer's side, once the S20_CREATE packet is received, a sanity check is performed on the capabilities. If all is well, the viewer replies with an S20_JOIN broadcast. At this point, the viewer and host are in a share. The caches are reset, a sync packet is sent to the creator, and the font list is sent. The viewer then receives the sync from the host, the font packet, and the desktop or application HET. The viewer then creates cursor cache, a palette cache, and a bitmap cache all of size the host said. The viewer then creates the compression saved history and the order decoding structures and memory. Finally, the viewer creates a bitmap for that person's desktop, or window to view what's shared. When the viewer wants to stop viewing, he sends a HET packet indicating that nothing is shared. The viewer then destroys the window, and frees the blocks shared. However, the viewer stays in the list.

Detailed Description Text (38):
For application sharing, the host needs to represent the window to the viewer, with a list of what is shared, the z-order, the shape and position of each window, and what is currently active. When the host is being controlled by a remote, he uses a control timer mechanism to ensure that other viewers' screens are kept up to date. Therefore, the playback of input first processes fast input such as the mouse clicks, cursor position and shape. Next, the window list for application sharing indicating what and where things are is processed. Finally, the graphics are processed, including the list of orders and screen data. If, for any reason, the window list or graphics cannot be sent, they are skipped so that the fast input may again be processed.

Detailed Description Text (49):
Basically, there is a pretty well-defined sequence of network traffic with control. The controller sends input events to the controlled person. The controlled person then broadcasts notifications to everybody in the conference of cursor position/shape changes and window position/appearance updates caused by playing back the input from the controller. Operation of this new control structure may be better understood from the following simple examples of how this new model works. Note that a host is a conference member sharing applications/desktop.

Detailed Description Text (60):
If the user turns off allowing control, queued request responses are all turned into "denied-control not allowed" responses. Also, request responses following a

queued "accept" must all be "denied" responses. If the user turns off allowing control, current controller is bounced as well. If the user tries to take control of a second host when a take control request is still queued, the first one is superseded. In other words, only one take control request will ever be queued. The user can release control of a host with a take control request queued. If that happens, the release simply cancels the queued take. If the user tries to take control of a host when an earlier take control request has been sent but has not yet been responded to, the first one is canceled by a "release" packet. In the meantime, the controller can do whatever it wants when waiting to hear back. A preferred embodiment of the instant invention includes a message box with a "cancel" button. Likewise, the host can do whatever it wants when it receives a "take control" request, provided it follows the rules. In a preferred embodiment, a message box with "person x would like to take control, ok/cancel" is displayed, and incoming requests are handled in order of receipt. Further, the system may, via SDK code, decide that the new controller is the one, bounce the current one, and allow the new user to take control. Further, as described above, it could remotely push control by asking a remote to take control of it.

Other Reference Publication (26):
IMTC, "IMTC Search: H.245," [web page], 1999. http://www.imtc.org/cgi-bin/query.cgi. [Accessed May 11, 1999].

Other Reference Publication (31):
"NetMeeting 3 Beta," [web page], Updated Mar. 23, 1998. http://www.microsoft.com/netmeeting/Beta30.htm. [Accessed May 7, 1999].

Other Reference Publication (38):
Microsoft Corporation, "How to Parse a Certificate Using Active Server Pages (ASP)," [web page], Updated Dec. 1996. http://msdn.microsoft.com/workshop/security/client/parse.asp. [Accessed May 20, 1999].

Other Reference Publication (39):
Microsoft Corporation, "Securing Data Transmissions with Secure Sockets Layer (SSL)," [web page], Updated Dec. 1996. http://msdn.microsoft.com/workshop/security/client/iis_ssl.asp. [Accessed May 20, 1999].

Other Reference Publication (55):
"NetMeeting 2.1 Features," [web page], Microsoft Corporation, Update Jan. 5, 1998. http://www.microsoft.com/netmeeting/features/main.htm. [Accessed May 4, 1999].

Other Reference Publication (56):
Microsoft Corporation, "Microsoft NetMeeting 2.0: Overview and Frequently Asked Questions," [web page], Updated Jul. 15, 1997. http://msdn.microsoft.com/library/backgrnd/html/msdn_netmofaq.htm. [Accessed May 4, 1999].

CLAIMS:

8. The method of claim 7, further comprising the steps of: passing control of the shared application to a conference participant; transmitting privately control commands for the shared application between the host and the conference participant in control of the shared application; and periodically broadcasting from the host updates to the shared application program to all of the conference participants.

9. The method of claim 8, further comprising the step of ignoring within the conference participant controlling the application program the broadcast update from the host.

10. The method of claim 8, wherein the step of periodically broadcasting updates includes the step of periodically broadcasting updates with a sync, further comprising the step of updating the shared program information within the conference participant controlling the application program.

11. The method of claim 1, further comprising the steps of: receiving a join request from a new member; broadcasting a respond packet from the host; adding the host to a member list within the new member in response to the respond packet from the host; adding the new member into a member list within each of the conference participants in response to the respond packet from the host; broadcasting a respond packet from each of the conference members; adding each of the conference members to the member list within the new member in response to the respond packet from each of the new members; and ignoring within each of the conference members the respond packets from each of the conference members already included in the member lists.

13. The method of claim 1, further comprising the steps of: receiving a leave broadcast packet from one of the conference. members; and deleting the conference member who broadcast the leave packet from a member list in each of the conference members.

14. The method of claim 1, further comprising the steps of: receiving a delete broadcast packet from the host identifying a conference member; and deleting the conference member identified in the delete packet from a member list in each of the conference members.

15. The method of claim 1, further comprising the steps of: receiving an end broadcast packet from the host; and deleting caches required by the host; and deleting a member list of all participants in the share.

17. The method of claim 1, further comprising the steps of: receiving a join request from a new member; broadcasting a respond packet from the host; adding the host to a member list within the new member in response to the respond packet from the host; adding the new member into a member list within each of the conference participants in response to the respond packet from the host; transmitting privately a respond packet from each of the conference members to the new member; and adding each of the conference members to the member list within the new member in response to the respond packet from each of the new members.

27. The method of claim 26, further comprising the steps of: receiving application program control inputs from the conference participant controlling the application program; and periodically broadcasting control input updates to the application program.

31. The method of claim 30, further comprising the steps of: receiving application program control inputs from the conference participant controlling the application program; and periodically broadcasting control input updates to the application program.

36. The method of claim 35, further comprising the steps of: receiving by the host application program control inputs from the conference participant controlling the application program; and periodically broadcasting by the host control input updates to the application program.

44. The method of claim 43, further comprising the steps of: receiving by the host application program control inputs from the conference participant controlling the application program; and periodically broadcasting by the host control input updates to the application program.